МИНОБРНАКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Хакасский государственный университет им. Н. Ф. Катанова»

Колледж педагогического образования, информатики и права

ПЦК естественнонаучных дисциплин, математики и информатики

РЕФЕРАТ

на тему:

Ядро операционной системы

Автор реферата: _____ О. И. Кидиеков
(подпись) (инициалы, фамилия)

Специальность: 230115 - Программирование в компьютерных системах

Курс: II

Группа: И-21

Зачет/незачет: _____

Руководитель: _____ О.П. Когумбаева
(подпись, дата) (инициалы, фамилия)

г. Абакан, 2018г.

Содержание:

Введение

Ядро́ — центральная часть операционной системы (ОС), обеспечивающая приложениям координированный доступ к ресурсам компьютера, таким как процессорное время, память, внешнее аппаратное обеспечение, внешнее устройство ввода и вывода информации, переводя команды языка приложений на язык двоичных кодов, которые понимает компьютер. Также обычно ядро предоставляет сервисы файловой системы и сетевых протоколов.

Как основополагающий элемент ОС, ядро представляет собой наиболее низкий уровень абстракции для доступа приложений к ресурсам системы, необходимым для их работы. Как правило, ядро предоставляет такой доступ исполняемым процессам соответствующих приложений за счёт использования механизмов межпроцессного взаимодействия и обращения приложений к системным вызовам ОС. Описанная задача может различаться в зависимости от типа архитектуры ядра и способа её реализации.

Целью данного реферата является исследование особенностей строения и способов комбинирования ядер операционной системы.

Необходимо выполнить ряд следующих задач для достижения цели:

1. Узнать что такое ядро операционной системы.
2. Рассмотреть виды ядер операционной системы.
3. Различные виды комбинаций ядер.
4. Выделить преимущества и недостатки этой комбинированных ядер в сравнении с простыми.

1. Типы архитектур ядер операционных систем

1.1. Монолитное ядро

Монолитное ядро предоставляет богатый набор абстракций оборудования. Все части монолитного ядра работают в одном адресном пространстве. Это такая схема операционной системы, при которой все компоненты её ядра являются составными частями одной программы, используют общие структуры данных и взаимодействуют друг с другом путём непосредственного вызова процедур. Монолитное ядро — старейший способ организации операционных систем. Примером систем с монолитным ядром является большинство UNIX-систем.

- *Достоинства*: Скорость работы, упрощённая разработка модулей[1].
- *Недостатки:* Поскольку всё ядро работает в одном адресном пространстве, сбой в одном из компонентов может нарушить работоспособность всей системы.

Примеры: Традиционные ядра UNIX (такие как BSD), Linux; ядро MS-DOS, ядро KolibriOS.

Некоторые старые монолитные ядра, в особенности систем класса UNIX/Linux, требовали перекомпиляции при любом изменении состава оборудования. Большинство современных ядер позволяют во время работы подгружать *модули*, выполняющие часть функций ядра. В этом случае компоненты операционной системы являются не самостоятельными модулями, а составными частями одной большой программы, называемой монолитным ядром (monolithic kernel), которое представляет собой набор процедур, каждая из которых может вызвать каждую. Все процедуры работают в привилегированном режиме.

1.2. Модульное ядро

Модульное ядро — современная, усовершенствованная модификация архитектуры монолитных ядер операционных систем.

В отличие от «классических» монолитных ядер, модульные ядра, как правило, не требуют полной перекомпиляции ядра при изменении состава

аппаратного обеспечения компьютера. Вместо этого модульные ядра предоставляют тот или иной механизм подгрузки модулей ядра, поддерживающих то или иное аппаратное обеспечение (например, драйверов). При этом подгрузка модулей может быть как динамической (выполняемой «на лету», без перезагрузки ОС, в работающей системе), так и статической (выполняемой при перезагрузке ОС после переконфигурирования системы на загрузку тех или иных модулей).

1.3. Микроядро

Микроядро предоставляет только элементарные функции управления процессами и минимальный набор абстракций для работы с оборудованием . Бóльшая часть работы осуществляется с помощью специальных пользовательских процессов, называемых *сервисами*. Решающим критерием «микроядерности» является размещение всех или почти всех драйверов и модулей в сервисных процессах, иногда с явной невозможностью загрузки любых модулей расширения в собственно микроядро, а также разработки таких расширений.

- *Достоинства:* Устойчивость к сбоям оборудования, ошибкам в компонентах системы. Основное достоинство микроядерной архитектуры — высокая степень модульности ядра операционной системы. Это существенно упрощает добавление в него новых компонентов. В микроядерной операционной системе можно, не прерывая её работы, загружать и выгружать новые драйверы, файловые системы и т. д. Существенно упрощается процесс отладки компонентов ядра, так как новая версия драйвера может загружаться без перезапуска всей операционной системы. Компоненты ядра операционной системы ничем принципиально не отличаются от пользовательских программ, поэтому для их отладки можно применять обычные средства. Микроядерная архитектура повышает надежность системы, поскольку ошибка на уровне непривилегированной программы менее опасна, чем отказ на уровне режима ядра.
- *Недостатки*: Передача данных между процессами требует накладных расходов.

Классические микроядра предоставляют лишь очень небольшой набор низкоуровневых примитивов, или системных вызовов, реализующих базовые сервисы операционной системы.

- Сервисные процессы (в принятой в семействе UNIX терминологии — «демоны») активно используются в самых различных ОС для задач типа запуска программ по расписанию (UNIX и Windows NT), ведения журналов событий (UNIX и Windows NT), централизованной проверки паролей и хранения пароля текущего интерактивного пользователя в специально ограниченной области памяти (Windows NT). Тем не менее, не следует считать ОС микроядерными только из-за использований такой архитектуры.

Примеры: Symbian OS; Windows CE; OpenVMS; Mach, используемый в GNU/Hurd и Mac OS X; QNX; AIX; Minix; ChorusOS; AmigaOS; MorphOS.

Примечание. Де́мон (англ. daemon) — в системах класса UNIX — программа, работающая в фоновом режиме без прямого общения с пользователем. Демоны обычно запускаются во время загрузки системы. Типичные задачи демонов: серверы сетевых протоколов (HTTP, FTP, электронная почта и др.), управление оборудованием, поддержка очередей печати, управление выполнением заданий по расписанию и т. д. В техническом смысле демоном считается процесс, который не имеет управляющего терминала. Чаще всего (но не обязательно) предком демона является init — корневой процесс UNIX. В системах Solaris 10 и OpenSolaris для управления демонами используется Service Management Facility. В системах Windows аналогичный класс программ называется «Службы» (англ. Services), которые, впрочем, тоже иногда называют демонами. Термин был придуман программистами проекта MAC Массачусетского технологического института, он отсылает к персонажу мысленного эксперимента, демону Максвелла, занимающегося сортировкой молекул в фоновом режиме. Системы UNIX унаследовали данную терминологию. Термин «даймон» (daemon, dæmon, греч. δαίμων) также относится к персонажам греческой мифологии, выполняющим задачи, за которые не хотят браться боги.

Как утверждается в «Справочнике системного администратора UNIX», в Древней Греции понятие «персональный даймон» было, отчасти, сопоставимо с современным понятием «ангел-хранитель». Иногда слово daemon интерпретируют как акроним англ. Disk and execution monitor. Операционные системы семейства BSD используют изображение демона в качестве логотипа. Классические микроядра предоставляют лишь очень небольшой набор низкоуровневых примитивов, или системных вызовов, реализующих базовые сервисы операционной системы. К ним относятся: управление адресным пространством оперативной памяти. управление адресным пространством виртуальной памяти. управление процессами и потоками (нитями). средства межпроцессной коммуникации. Все остальные сервисы ОС, в классических монолитных ядрах предоставляемые непосредственно ядром, в микроядерных архитектурах реализуются в адресном пространстве пользователя (Ring3) и называются сервисами. Примерами таких сервисов, выносимых в пространство пользователя в микроядерных архитектурах, являются сетевые сервисы, файловая система, драйверы. Такая конструкция позволяет улучшить общее быстродействие системы (небольшое микроядро может уместиться в кэше процессора). Основное достоинство микроядерной архитектуры — высокая степень модульности ядра операционной системы. Это существенно упрощает добавление в него новых компонентов. В микроядерной операционной системе можно, не прерывая ее работы, загружать и выгружать новые драйверы, файловые системы и т. д. Существенно упрощается процесс отладки компонентов ядра, так как новая версия драйвера может загружаться без перезапуска всей операционной системы. Компоненты ядра операционной системы ничем принципиально не отличаются от пользовательских программ, поэтому для их отладки можно применять обычные средства. Микроядерная архитектура повышает надежность системы, поскольку ошибка на уровне непривилегированной программы менее опасна, чем отказ на уровне режима ядра. И чтобы добавить в ОС с микроядром драйвер того или иного устройства, не надо перекомпилировать всё ядро, а надо лишь отдельно откомпилировать этот драйвер и запустить его в пользовательском

пространстве. В то же время микроядерная архитектура операционной системы вносит дополнительные накладные расходы, связанные с передачей сообщений, что отрицательно влияет на производительность. Для того чтобы микроядерная операционная система по скорости не уступала операционным системам на базе монолитного ядра, требуется очень аккуратно проектировать разбиение системы на компоненты, стараясь минимизировать взаимодействие между ними. Таким образом, основная сложность при создании микроядерных операционных систем — необходимость очень аккуратного проектирования. Микроядра типа ядра ОС Minix и GNU Hurd развиваются медленно, гораздо медленнее, чем Linux и ядро систем семейства BSD. По словам создателя Minix3, Таненбаума, он «пытается построить сверхнадёжную (very highly reliable) систему. Она может использоваться в том числе на серверах, которым необходимы годы безотказной работы».[1] Классическим примером микроядерной системы является Symbian OS. Это пример распространенной и отработанной микроядерной (а начиная с версии Symbian OS v8.1, и наноядерной) операционной системы. В отличие от Windows NT создателям Symbian OS удалось совместить эффективность и концептуальную стройность, несмотря на то что современные версии этой системы предоставляют обширные возможности, в том числе средства для работы с потоковыми данными, стеками протоколов, критичными к латентности ядра, графикой и видео высокого разрешения). Разработчики Symbian вынесли практически все прикладные (т.е. выходящие за пределы компетенции ядра) задачи в модули-серверы, функционирующие в пользовательском адресном пространстве. В ОС Windows NT версий 3.x микроядерная архитектура с сервисным процессом использовалась для подсистемы графики и пользовательского интерфейса. В частности, драйвер графической аппаратуры загружался в контекст сервисного процесса, а не ядра. Начиная с версии 4, от этого отказались, сервисный процесс сохранился только для управления консольными окнами командной строки, а собственно графическая подсистема вместе с драйвером аппаратуры (в том числе трехмерной графики) переместилась в специально обособленный регион ядра ОС. ОС Windows CE (и созданные на ее

основе сборки, такие, как Windows Mobile), будучи практически полностью совместимой (как подмножество) с Windows NT по вызовам и методам программирования приложений, тем не менее полностью отличается от Windows NT по внутренней архитектуре и является микроядерной ОС с выносом всех драйверов устройств, сетевых стеков и графической подсистемы в сервисные процессы. Недостаток — плата за принудительное «переключение» процессов в ядре (переключение контекста); этот факт собственно и объясняет трудности в проектировании и написании ядер подобной конструкции. Эти недостатки способны обойти ОС, использующие архитектуру экзоядра, являющуюся дальнейшим развитием микроядерной архитектуры.

1.4. Экзоядро

Экзоядро — ядро операционной системы, предоставляющее лишь функции для взаимодействия между процессами, безопасного выделения и освобождения ресурсов. Предполагается, что API для прикладных программ будут предоставляться внешними по отношению к ядру библиотеками (откуда и название архитектуры).

В традиционных операционных системах ядро предоставляет не только минимальный набор сервисов, обеспечивающих выполнение программ, но и большое количество высокоуровневых абстракций для использования разнородных ресурсов компьютера: оперативной памяти, жёстких дисков, сетевых подключений. В отличие от них, ОС на основе экзоядра предоставляет лишь набор сервисов для взаимодействия между приложениями, а также необходимый минимум функций, связанных с защитой: выделение и высвобождение ресурсов, контроль прав доступа, и т. д. Экзоядро не занимается предоставлением абстракций для физических ресурсов — эти функции выносятся в библиотеку пользовательского уровня (так называемую libOS).

Основная идея операционной системы на основе экзоядра состоит в том, что ядро должно выполнять лишь функции координатора для небольших процессов, связанных только одним ограничением — экзоядро должно иметь возможность гарантировать безопасное выделение и освобождение ресурсов оборудования. В

отличие от ОС на основе микроядра, ОС, базирующиеся на экзоядре, обеспечивают гораздо большую эффективность за счет отсутствия необходимости в переключении между процессами при каждом обращении к оборудованию.

Архитектуры на основе экзоядер являются дальнейшим развитием и усовершенствованием микроядерных архитектур и одновременно ужесточают требования к минималистичности и простоте кода ядра.

libOS может обеспечивать произвольный набор абстракций, совместимый с той или иной уже существующей операционной системой, например Linux или Windows.

1.5. Наноядро

Наноядро — архитектура ядра операционной системы, в рамках которой крайне упрощённое и минималистичное ядро выполняет лишь одну задачу — обработку аппаратных прерываний, генерируемых устройствами компьютера. После обработки прерываний от аппаратуры наноядро, в свою очередь, посылает информацию о результатах обработки (например, полученные с клавиатуры символы) вышележащему программному обеспечению при помощи того же механизма прерываний. Также часто реализуют минимальную поддержку потоков: создание и переключение. Наиболее часто в современных компьютерах наноядра используются для виртуализации аппаратного обеспечения реальных компьютеров или для реализации механизма гипервизора, с целью позволить нескольким или многим различным операционным системам работать одновременно и параллельно на одном и том же компьютере. Наноядра также могут использоваться для обеспечения переносимости операционных систем на разное аппаратное обеспечение или для обеспечения возможности запуска «старой» операционной системы на новом, несовместимом аппаратном обеспечении без её полного переписывания и портирования. Наноядро может быть настолько маленьким и примитивным, что даже важнейшие устройства, находящиеся непосредственно на материнской плате или на плате контроллера встраиваемого устройства, такие, как таймер или программируемый контроллер

прерываний, обслуживаются специальными драйверами устройств, а не непосредственно ядром. Такого рода сверхминималистичные наноядра называют иногда пикоядрами. Примером является KeyKOS — самая первая ОС на наноядре. Первая версия вышла ещё в 1983-м году.

1.6. Гибридное ядро

Гибридные ядра — это модифицированные микроядра, позволяющие для ускорения работы запускать «несущественные» части в пространстве ядра. Пример: ядра ОС Microsoft Windows семейства NT: Windows NT, Windows 2000, Windows XP, Windows Vista, Windows 7. Все рассмотренные подходы к построению операционных систем имеют свои достоинства и недостатки. В большинстве случаев современные операционные системы используют различные комбинации этих подходов. Так, например сейчас, ядро «Linux» представляет собой монолитную систему с отдельными элементами модульного ядра. При компиляции ядра можно разрешить динамическую загрузку и выгрузку очень многих компонентов ядра — так называемых модулей. В момент загрузки модуля его код загружается на уровне системы и связывается с остальной частью ядра. Внутри модуля могут использоваться любые экспортируемые ядром функции.

Существуют варианты ОС GNU (Debian GNU/Hurd), в которых вместо монолитного ядра применяется ядро Mach (такое же, как в Hurd), а поверх него в пользовательском пространстве работают те же самые процессы, которые при использовании Linux были бы частью ядра. Другим примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра. Так устроены 4.4BSD и MkLinux, основанные на микроядре Mach. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. Все остальные функции, в том числе взаимодействие с прикладными программами, осуществляется монолитным ядром. Данный подход сформировался в результате попыток использовать преимущества микроядерной архитектуры, сохраняя по возможности хорошо отлаженный код монолитного ядра.

2. Комбинация разных подходов

Все рассмотренные подходы к построению операционных систем имеют свои достоинства и недостатки. В большинстве случаев современные операционные системы используют различные комбинации этих подходов. Так, например, сейчас ядро «Linux» представляет собой монолитную систему с отдельными элементами модульного ядра. При компиляции ядра можно разрешить динамическую загрузку и выгрузку очень многих компонентов ядра — так называемых модулей. В момент загрузки модуля его код загружается на уровне системы и связывается с остальной частью ядра. Внутри модуля могут использоваться любые экспортируемые ядром функции.

Существуют варианты ОС GNU, в которых вместо монолитного ядра применяется ядро Mach (такое же, как в Hurd), а поверх него крутятся в пользовательском пространстве те же самые процессы, которые при использовании Linux были бы частью ядра. Другим примером смешанного подхода может служить возможность запуска операционной системы с монолитным ядром под управлением микроядра. Так устроены 4.4BSD и MkLinux, основанные на микроядре Mach. Микроядро обеспечивает управление виртуальной памятью и работу низкоуровневых драйверов. Все остальные функции, в том числе взаимодействие с прикладными программами, осуществляются монолитным ядром. Данный подход сформировался в результате попыток использовать преимущества микроядерной архитектуры, сохраняя по возможности хорошо отлаженный код монолитного ядра.

Смешанное ядро, в принципе, должно объединять преимущества монолитного ядра и микроядра: казалось бы, микроядро и монолитное ядро — крайности, а смешанное — золотая середина. В них возможно добавлять драйверы устройств двумя способами: и внутрь ядра, и в пользовательское пространство. Но на практике концепция смешанного ядра часто подчёркивает не только достоинства, но и недостатки обоих типов ядер.

Заключение

Целью реферата было исследование преимуществ и недостатков различных типов ядер для операционной системы. Был поставлен ряд задач, которые необходимо было выполнить, для достижения намеченной цели. Если рассмотреть последовательно каждый пункт, то можно сделать вывод, что цель реферата достигнута: дан развернутый ответ на вопрос, что такое ядро операционной системы; рассмотрены виды ядер операционной системы: выявлены основные преимущества и недостатки комбинированных ядер; сделаны соответствующие выводы, посредством систематизации и анализа полученных данных.

На данный момент используются различные виды ядер в различных операционных системах. На различных ядрах и их комбинациях, проектируются новые Операционные Системы, как с закрытым кодом, так и с открытым.

Данная тема действительно актуальна в нынешнее время, так как большинство разрабатываемых Операционных Систем в нынешнее время строится на комбинированных ядрах. С помощью данного подхода разработчики стараются минимизировать недостатки и получить максимум плюсов от разных типов ядер Операционной системы.

Библиографический список

1. Астахова, И.Ф. Компьютерные науки. Деревья, операционные системы, сети / И.Ф. Астахова, И.К. Астанин и др. - М.: Физматлит, 2013. - 88 с.
2. Астахова, И.Ф. Компьютерные науки. Деревья, операционные системы, сети / И.Ф. Астахова и др. - М.: Физматлит, 2013. - 88 с.
3. Дейтел, Х., М. Операционные системы. Основы и принципы. Т. 1 / Х. М. Дейтел, Д.Р. Чофнес. - М.: Бином, 2016. - 1024 с.
4. Дейтел, Х.М. Операционные системы. Распределенные системы, сети, безопасность / Х.М. Дейтел, Д.Р. Чофнес. - М.: Бином, 2013. - 704 с.
5. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. - СПб.: Питер, 2013. - 1120 с.
6. Введение в программные системы и их разработку // Национальный Открытый Университет «ИНТУИТ» • 2016 год • 650 страниц [Электронный ресурс]. URL: http://www.knigafund.ru/books/177972 (дата обращения: 13.01.2018).
7. Ядро (Операционные Системы) // Материал из Национальной библиотеки им. Н. Э. Баумана. Последнее изменение этой страницы: 15:57, 24 августа 2017. [Электронный ресурс]. URL: https://ru.bmstu.wiki/Ядро_(Операционные_Системы) (дата обращения: 13.01.2018).
8. Операционная система Материал из Национальной библиотеки им. Н. Э. Баумана. Последнее изменение этой страницы: 15:54, 24 августа 2017. [Электронный ресурс]. URL: https://ru.bmstu.wiki/Операционная_система (дата обращения: 13.01.2018).
9. Микроядро // Материал из Национальной библиотеки им. Н. Э. Баумана. Последнее изменение этой страницы: 16:19, 24 августа 2017. [Электронный ресурс]. URL: https://ru.bmstu.wiki/Микроядро (дата обращения: 13.01.2018).
10. Ефремов,Денис Валентинович. Конструирование ядра операционной системы : учеб. пособие / Д. В. Ефремов, Н. Ю. Комаров, А. В. Хорошилов ;

Моск. гос. ун-т им. М. В. Ломоносова, Фак. вычисл. математики и кибернетики. М.: МАКС Пресс : Изд. отд. фак. ВМиК МГУ им. М. В. Ломоносова, 2015